SILICON UPDATE

Tom Cantrell

Working the 'Net

Shouldn't every engineer know a little more about the process of network-enabling his or her widget? Tom definitely thinks so. With Tom on your side, you'll be working the 'Net to your advantage in no time.

Word comes down from on high: "We've got to network-enable our widget!"

As a design engineer bombarded by management mandates, you may sometimes feel your job is not to reason why, only to do or die. But, when it comes to getting embedded applications on the I-way, believe me when I say that you (and everybody else involved) had better understand why, or you won't know what to do. Worse yet, you'll do something you'll be sorry you did.

The fact is, your first response needs to be, "Er, what exactly do you mean by network-enable?" That should give everybody something to chew on. Let's try to come up with a few answers as we take a look at a gaggle of gadgets that, each in their own way, bring the 'Net to embedded apps (see Photo 1).

ETHER OR NOT

Over the last decade, most corporations have had to migrate from whatever proprietary, ad-hoc networking lash-up they had before, some of which were schemes going as far back as the Bronze Age of the mainframe and minicomputers. Big-iron legacy applications and hardware die hard, so the transition typically doesn't happen overnight.

It's easier to change the wire than the proprietary software and 24/7 devices behind it. In the course of enterprise IT reconfiguration, it's not hard to imagine situations in which there is a need to maintain some proprietary RS-232-based hardware and software across the transition to TCP/IP and Ethernet.

Consider an existing application with a factory-floor, RS-232 device—perhaps an electronic scale, ticket printer, or bar code reader—hard-wired to a PC. Having run Ethernet cable hither and yon, now what? Neither the existing application software running on the PC nor the RS-232 device knows anything about Ethernet, not to mention the Internet.

The solution is conceptually simple: turn the Internet into a virtual RS-232 cable with products like the Lantronix CoBox or i2Chip IGM7100. In essence, they're RS-232-to-Ethernet converters. Connect one of them to the COM port on your PC and one to the RS-232 port on your embedded device. Now, you've got Ethernet jacks at each end, and you can plug in an Ethernet cable, or for that matter, the entire Internet in-between them.

The beauty of the scheme is that it's transparent to the application hardware and software at each end. Both the PC and embedded device continue to talk to their UARTs, blissfully unaware that traffic is actually moving over the network.

You can cut back to just one adapter if the PC has a built-in Ethernet port, which will be the most likely case given the entire scenario revolves around upgrading to a network. Normally that would call for rewriting the software on the PC to talk to the Ethernet instead of the COM port. An easier approach is to use COM port redirector software that installs a driver to trap COM port-related OS calls and routes them via the Ethernet port.

That's the approach I took, using the redirector software that came with the Lantronix box. Sure enough, I was able to connect an existing RS-232-based application to both the Lantronix and

i2Chip boards. Everything worked just as advertised (i.e., exactly as before without software changes).

There is one thing to watch out for, though. The serial port on both of these converters is designed to plug directly into a PC to provide an alternate path (instead of Ethernet) for downloading and configuration. But, that means when you unplug your embedded device from the PC's COM port and plug it into the converter's serial port, you have to perform the old DTE-DCE switcheroo (i.e., swap the TX and RX pins). Don't toss your RS-232 breakout box just yet. The more things change the more they stay the same.

WEB SITE LITE

I suspect that when the marketing department bangs on the table for networking, they usually mean to say, "web browser enabling." In other words, the ability to interrogate, control, upgrade, and otherwise interact with an embedded product using a standard web browser.

These days, practically all embedded products have a user interface, whether it's a full-featured, touch-screen display or a DIP switch and blinking LED. The user interface even can be a PC with some kind of software making the connection, ranging from a simple terminal program to a custom-programmed GUI.

The advantage of the web-based approach is compelling. In short, it puts the user-interface ball squarely in the user's court, which turns out to be a win-win for everyone. Users get to choose their favorite browsing platform, which is typically a PC but is inexorably expanding to include PDAs, smart phones, web TVs, handheld games, and ultimately just about anything with a display and input device.

Meanwhile, the equipment supplier is relieved of the burden of having to make provisions for (and support) proprietary software and hardware at the user end. The engineering spec is simple: add an Ethernet jack and cough up a web page. All that's left is to figure out the simplest, quickest, and cheapest way to do it.

NetMedia targets this scenario with their SitePlayer, which is notable for its laser-like focus. All nonessential baggage is tossed overboard in pursuit of the singular goal of talking to a web browser. There's no socket-level programming, e-mail, execution of user code (although the web site alludes to a programming language in the works), or anything else. Thanks to it's single-minded purpose, SitePlayer gets by with a remarkably tiny bill of materials. It's literally made up of two chips: a standard Philips flash memory '51 and a commodity RealTek Ethernet MAC chip.

That's not to say such a streamlined approach doesn't come with some compromises. For example, because your SitePlayer-based web site is limited to the '51's 48-KB on-chip flash memory, leave your bloated eye candy at home. Nevertheless, with care you can come up with up with an attractive user interface that doesn't bust the bit budget (see Photo 2).

Of course, serving up a mere static web page isn't that hard nor is it very useful. You'll generally want to display or input dynamic data. For instance, a web-enabled thermostat would allow the user to enter a desired temperature as well as display the actual current temperature.

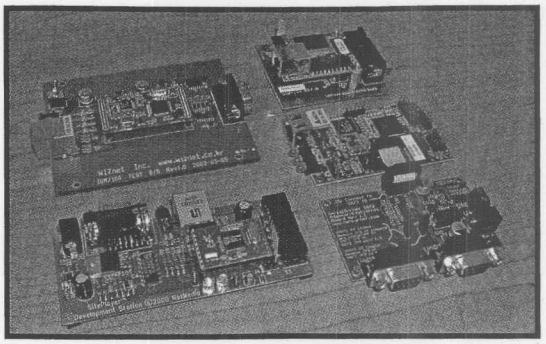

Photo 1—*Moving clockwise from the top left, you can see the i2Chip IGM7100, Lantronix CoBox Micro, NetBurner SB72, and SitePlayer. These are evaluation setups with the base modules plugged into carrier boards that include the RS-232 level shifter and connector, and, in the case of i2Chip and SitePlayer, the Ethernet PHY and connector.*

SitePlayer makes an otherwise static page dynamic by embedding special tags in the HTML. The tags, denoted by a caret (^), refer to variables that pass one-way or the other (or both ways) between the SitePlayer web server and your embedded device server. For instance, in the thermostat example, your web page could simply display the following string:

```
The current temperature is
 ^temp
```

As SitePlayer dishes up the web page, it will notice the ^temp variable and substitute the current value of the temperature variable.

SitePlayer knows about the ^temp variable by virtue of a SitePlayer definition file (extension .spd) that you create. Along with various configuration parameters (i.e., the IP address, whether or not to use DHCP or a password, etc.), the .spd file defines the location and data type for each variable.

On the device server side (i.e., RS-232), the embedded system (e.g., thermostat) updates the ^temp variable using a simple protocol (command and data) that accesses SitePlayer's internal memory. In the thermostat example, you would send the current temperature via the RS-232 to the location in SitePlayer cor-

Photo 2—*If you keep your design tight, you can get SitePlayer to serve up an appealing interface, even with only 48 KB of flash memory to play with.*

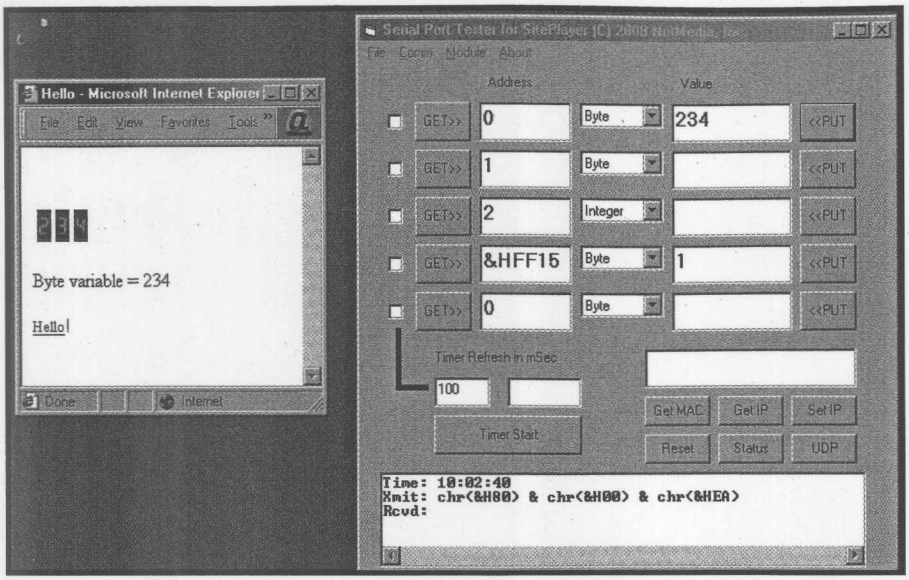

Photo 3—*A key SitePlayer technique is to parse the numeric value of a variable to create a file name that references a graphic file, in this case a seven-segment LED. Meanwhile, the Serial Tester utility facilitates PC-based debugging by allowing easy access to variables and monitoring the associated RS-232 command and response packets.*

responding to the `^temp` variable as defined in the .spd file. Now, the next time the page is refreshed the current temperature will be displayed.

Of course, as soon as you've crafted a simple character-oriented interface, the call will come down from on high to spiff it up with graphics for a more glamorous first impression. The SitePlayer allows you to, for example, replace a simple numeric variable with its virtual seven-segment LED equivalent. Photo 3 shows a web page in which `var` is defined as a byte displayed as a decimal number of up to three digits (i.e., 0–255).

SitePlayer allows you to pick off individual digits using a colon notation (`:`). Examples include `^var:3`, `^var:2`, and `^var:1`. In turn, the digits are grafted onto the file name calling out a particular LED image file. For instance, if the value of the variable is 234, `^var:1_LED.jpg` will bring in the file 4_LED.jpg, which is the bitmap image of an LED showing the number four. A variety of useful image files (e.g., seven-segment LEDs, sliders, and knobs) come with the SitePlayer software.

As usual, the complexity (if not the devil) is in the details. Using Word as my web page authoring tool, I couldn't find a way to generate the colon syntax. Thus, I had to edit the HTML directly, something I imagine is required for some of the other fancy constructs as well. Also, I was unable to come up with a scheme to blank leading LED zeros. There may be a way for users to kludge around it, but I would say this is a feature that should be added to the tools.

COM SPI WITH ME

Besides user-defined objects like variables, SitePlayer offers a number of built-in objects that you can access. These include various '51 resources such as the on-chip special function registers and I/O lines along with higher-level objects like COM and SPI. The former provides easy access to the '51's UART. For instance, referring back to Photo 3, clicking on `Hello` invokes a link that looks like:

```
<A HREF="comtest.spi?com=
 Hello%20World%0d%0a"
 >Hello</A>
```

which sends the `Hello World` string followed by a carriage return and line feed out the UART.

Meanwhile, the SPI object provides, as you might have guessed, access to SPI peripheral ICs. When using the SPI object, three of the eight available SitePlayer '51 I/O lines take on the clocked serial role (i.e., SCK, MOSI, and MISO) with any or all of the remaining five lines used as SPI chip selects. The manual contains examples of the admittedly cryptic HTML needed to perform a SPI access, but also explains that you may have to get your hands dirty (e.g., Java) to make much use of the cascade of bits associated with a particular peripheral chip.

There's even a universal datagram protocol (UDP) object that allows your device to send, receive, and broadcast packets to other UDP-capable devices, including, of course, other SitePlayers. Just remember that unlike TCP/IP, with UDP there are few guarantees in terms of delivery and correctness, so you'll have to harden your application against the possibility of network errors.

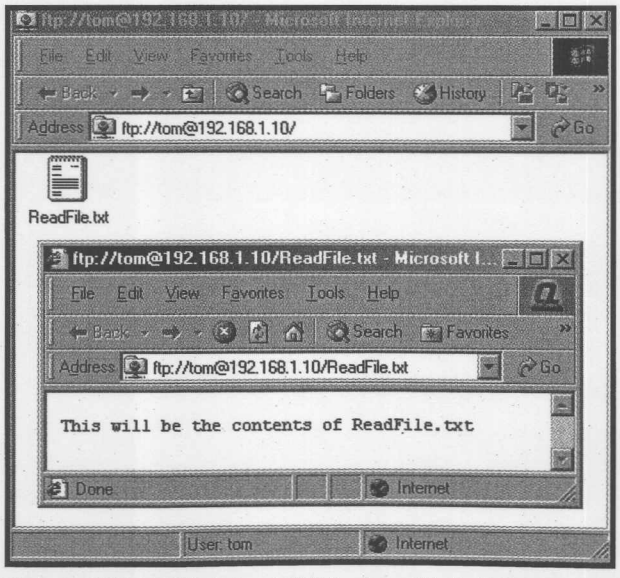

Photo 4—*With a TCP/IP stack and GNU development tools, NetBurner can do it all. For example, turning your embedded gadget into an FTP server provides an easy way to move application data to or from a file. Just copy and paste.*

FRESHEN UP

The SitePlayer package is reasonably friendly and easy to use, all the more so by virtue of some handy utilities that streamline development. The standard flow is to define your variables and such in the aforementioned SitePlayer definition file (.spd) and then create your web page using a combination of web-authoring tools and tweaking the result-

ing HTML. Put the HTML and all other files associated with your SitePlayer web site (e.g., the seven-segment LED .jpg files and any other graphics) in a folder and call up the SiteLinker program.

SiteLinker chews on your web site folder and spits out a SitePlayer binary (.spb) file that's downloaded into the '51's flash memory via the network. Refresh your web browser to see the results and repeat the cycle as necessary.

As I mentioned earlier, SitePlayer web sites are limited to 48 KB by the confines of the '51's on-chip flash memory. That means you have to be careful about going overboard with fancy graphics, long file names, and so on. The good news is that the 48-KB limit translates into a reasonably quick link-download-reflash cycle that takes only 20 to 30 s.

Nevertheless, even a relatively minor delay inhibits the quick iteration that's called for by the artistic exercise of creating a web page, not to mention the tweak-and-try personality of the SitePlayer tools. To that end, the SitePlayerPC utility provided essentially turns the PC into a big SitePlayer, allowing you to run your web site .spd binary file in situ.

Similarly, the serial tester utility allows development to proceed using the PC COM port as a stand-in for the embedded device that ultimately will be connected to the SitePlayer's serial port. The utility allows easy access (GET and PUT) to SitePlayer objects. It also displays the commands and responses traveling across the serial link, which is valuable assistance when you're writing the serial interface code that runs on your own device.

BURNING DOWN THE 'NET

The previously described products are all designed to ease the process of migrating an existing application toward a network-based solution. Eventually, or if you're starting from scratch, the time will come to consider consolidating your embedded application processing and networking to run on a single processor.

Enter NetBurner. Despite it's small size (and price), don't be fooled. Based on a 66-MHz Motorola MC5272 ColdFire chip (see Figure 1), NetBurner is a stand-alone 32-bit SBC with up to 512 KB of flash memory and a whopping 8 MB of SDRAM. Under the hood is a full-blown RTOS (µC/OS) and complete TCP/IP stack. Here, you'll find every letter in the alphabet soup of protocols that handle web serving, e-mail, file transfer (i.e., FTP), and all the rest.

What separates NetBurner from many other 32-bit SBCs are specific utilities, libraries, and application examples that tailor the device for quick and easy (or at least quicker and easier) Internet apps. Tools start with the familiar GNU toolchain, including a C compiler, assembler, debugger, linker, and so on. However, those of you who have dabbled with GNU before know that merely manhandling the tools, not to mention getting any work done, is a nontrivial task.

To make life easier, NetBurner comes with a predefined `make` script that handles the messy details. Notably, this includes consolidating

and compressing all related application and web files for packing into the flash memory. Then, after reset, NetBurner decompresses and transfers the code to SDRAM, which, in conjunction with the '5272's high-speed on-chip cache and RAM, delivers speedy processing.

For dynamic web pages, feel free to use all of the highfalutin techniques—such as Java applets, common gateway interface (CGI), and socket-level programming—that this 32-bit processor and its megs of memory can support. Just remember that even though the platform is downsized, such big-iron programming isn't for beginners or the faint of heart.

NetBurner offers an easier way out with a "tagged HTML" approach similar to the one I described earlier for SitePlayer. The difference is that the NetBurner tags, rather than just displaying variables and objects, perform a C function call for unlimited processing capability (see Listing 1). A particular item that appears on the web browser's screen might have 10 or 10,000 lines of code behind it.

I was impressed with the collection of nearly two-dozen example applications. They encompass everything from the simplest Hello World demo to e-mail, Telnet, PPP, UDP, FTP, and much more (see Photo 4). There's just no better way to learn how to do something than to see how it's done, and the examples will prove to be a helpful starting point for your own application.

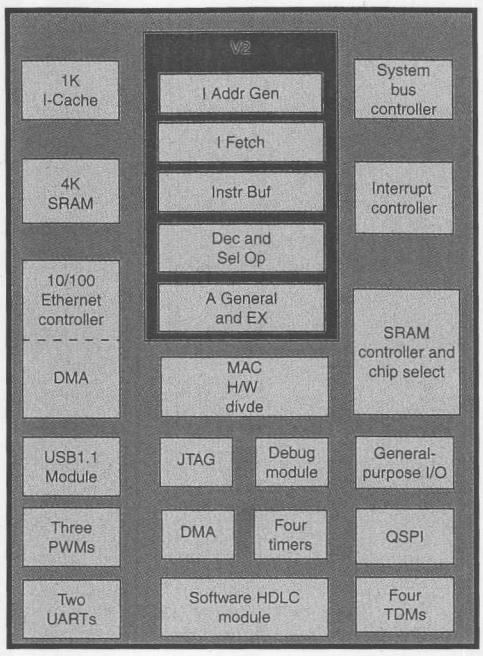

Figure 1—*Where's the kitchen sink? The Motorola MC5272 at the heart of NetBurner is a networking workhorse with built-in Ethernet, USB, SPI, and UARTs.*

HAVE IT YOUR WAY

Having seen the variety of choices, I think you can understand that there's more to getting your application on the I-way than meets the eye. The question of whether or not to do it is the easiest to answer—a resounding "yes" in my opinion—but only the first.

Perhaps you've got a legacy serial port app and simply want to change the wire with no intention of upgrading the hardware or software beyond that. If so, Ethernet enabling is the way to go with a simple converter such as the Lantronix CoBox or i2Chip IGM7100.

If you just want to put a web-friendly face in front of your existing hardware with minimal software changes, web browser enabling is really what you're talking about. The NetMedia SitePlayer is designed to do just that. And although I didn't get into it this time, note that easy e-mail enabling is also an option with products from the likes of ConnectOne and Cermetek.

Maybe you're starting from scratch and need a full-blown, Internet-enabled solution that can handle any and all protocols (e.g., web, e-mail, and FTP) as well as your application processing. That calls for a full-blown computing platform (à la NetBurner) with sufficient memory and MIPs to carry the load.

Yes, there are a lot of questions to work through, but take heart. The good news is that for every question you come up with there are folks working hard to make sure you can find just the right answer.

Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.

Listing 1—*NetBurner's approach to dynamic web pages embeds tagged C function calls in the HTML.*

```
<html>
<B>Click on the boxes <br>
to toggle the LEDs on the NetBurner board
<table border=1>
<tr>
<!--FUNCTIONCALL DoLeds -->
</tr>
</table>
<br>
Dip Switches:
<!--FUNCTIONCALL DoSwitches -->
</body>
</html>
```

SOURCES

IGM7100 Serial-to-Ethernet gateway module
i2Chip, Inc.
(408) 432-5081
www.i2chip.com

CoBox Micro device server
Lantronix, Inc.
(949) 453-3990
www.lantronix.com

MC5272 ColdFire Microprocessor
Motorola, Inc.
(847) 576-5000
www.motorola.com

SB72 Serial-to-Ethernet device
NetBurner, Inc.
(858) 558-0293
www.netburner.com

SitePlayer Ethernet web server
NetMedia, Inc.
(520) 544-4567
www.siteplayer.com